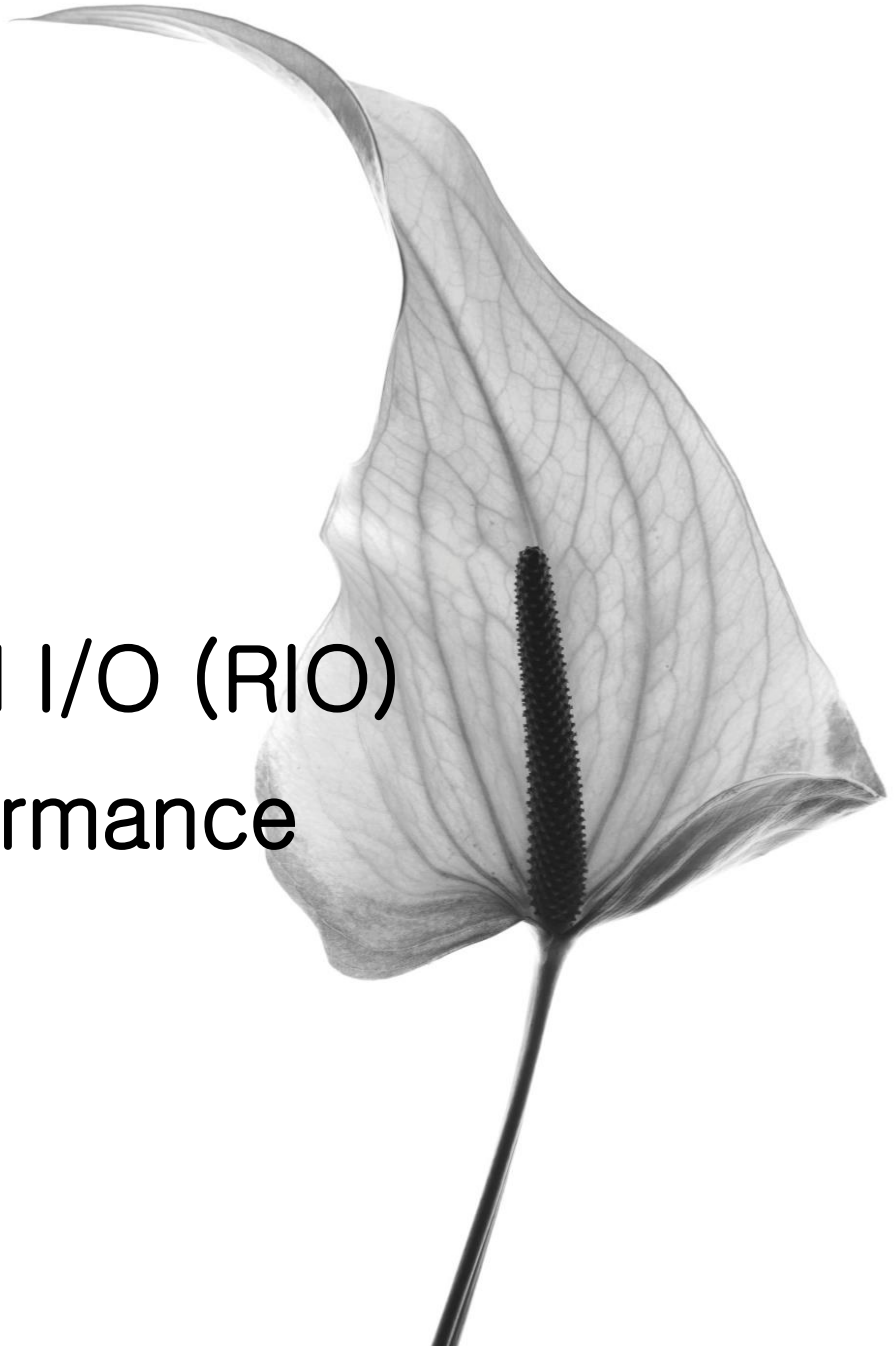
NHN INSTITUTE FOR THE
NEXT NETWORK



Windows Registered I/O (RIO) Introduction & Performance

Seungmo Koo (@sm9kr)

대한민국 온라인 게임 서버 제작자 모임

Agenda

NHN
NEXT

- IOCP: I/O Completion Port
- RIO: Registered I/O Network Extensions
- 성능 평가
- 결론
- Source Code Sample

IOCP 개요

NHN NEXT

- 다들 잘 아시는 것
  - Proactor 방식의 고성능 I/O Notification Model
    - 비동기 I/O 지원
  - Windows OS가 직접 효율적인 쓰레드 풀링 제공
    - context-switching을 줄이는 효과
  - Overlapped I/O 지원
    - 커널영역과 유저영역의 버퍼 공유 (memory page-locking)

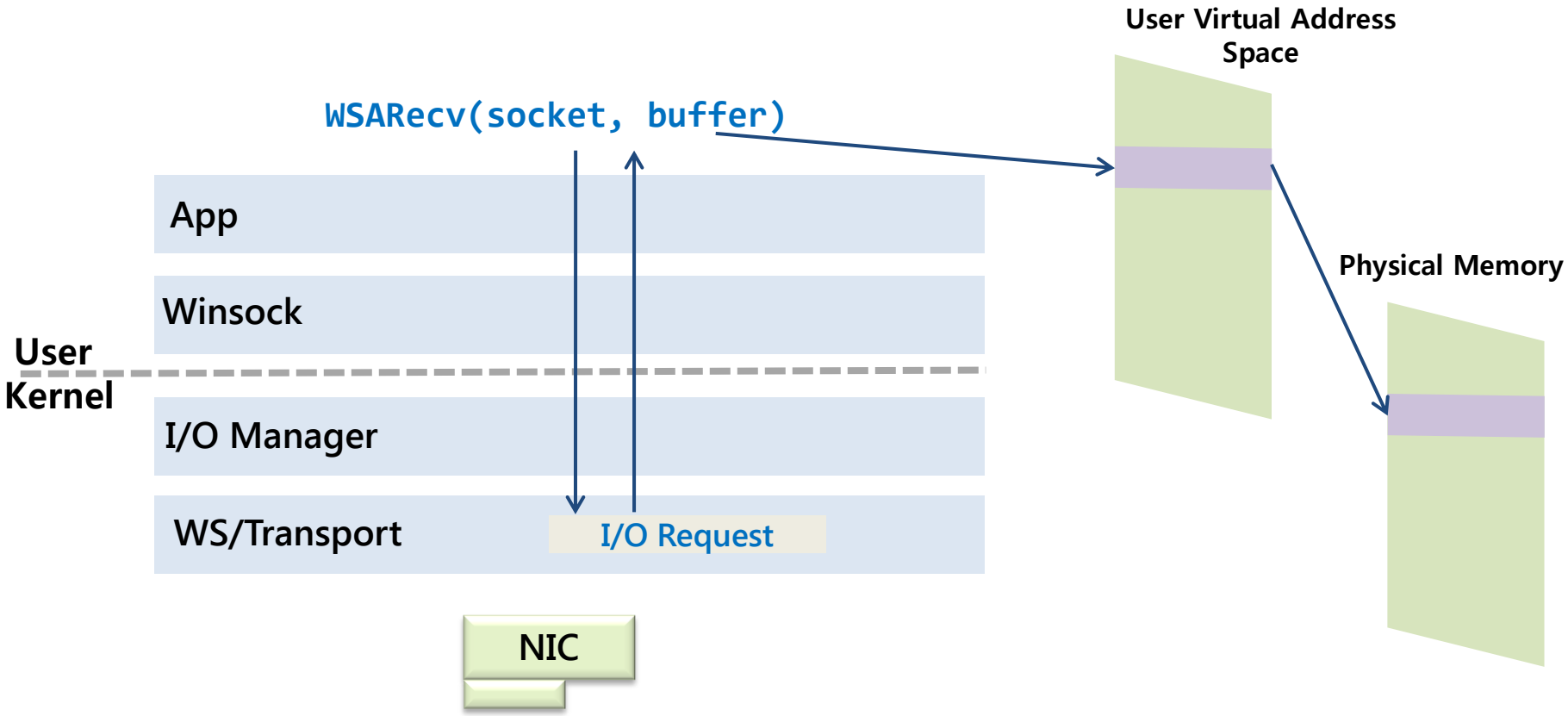
- 그럼에도 불구하고,
  - 하나의 I/O operation마다 버퍼 영역에 대한 page-lock/unlock
    - 특정 메모리에 대한 Pin/Unpin은 많은 CPU cycle요구
    - 그래서 RECV를 posting 할 때, page-locking을 피하여 CPU cycle을 줄이기 위해 zero-byte recv 꼼수를 사용해왔음
  - 하나의 I/O operation마다 시스템콜 호출
    - 유저모드-커널모드 전환 발생

IOCP 동작구조

NHN NEXT

- 기본적인 처리 흐름
  - I/O initiation → I/O processing → I/O completion

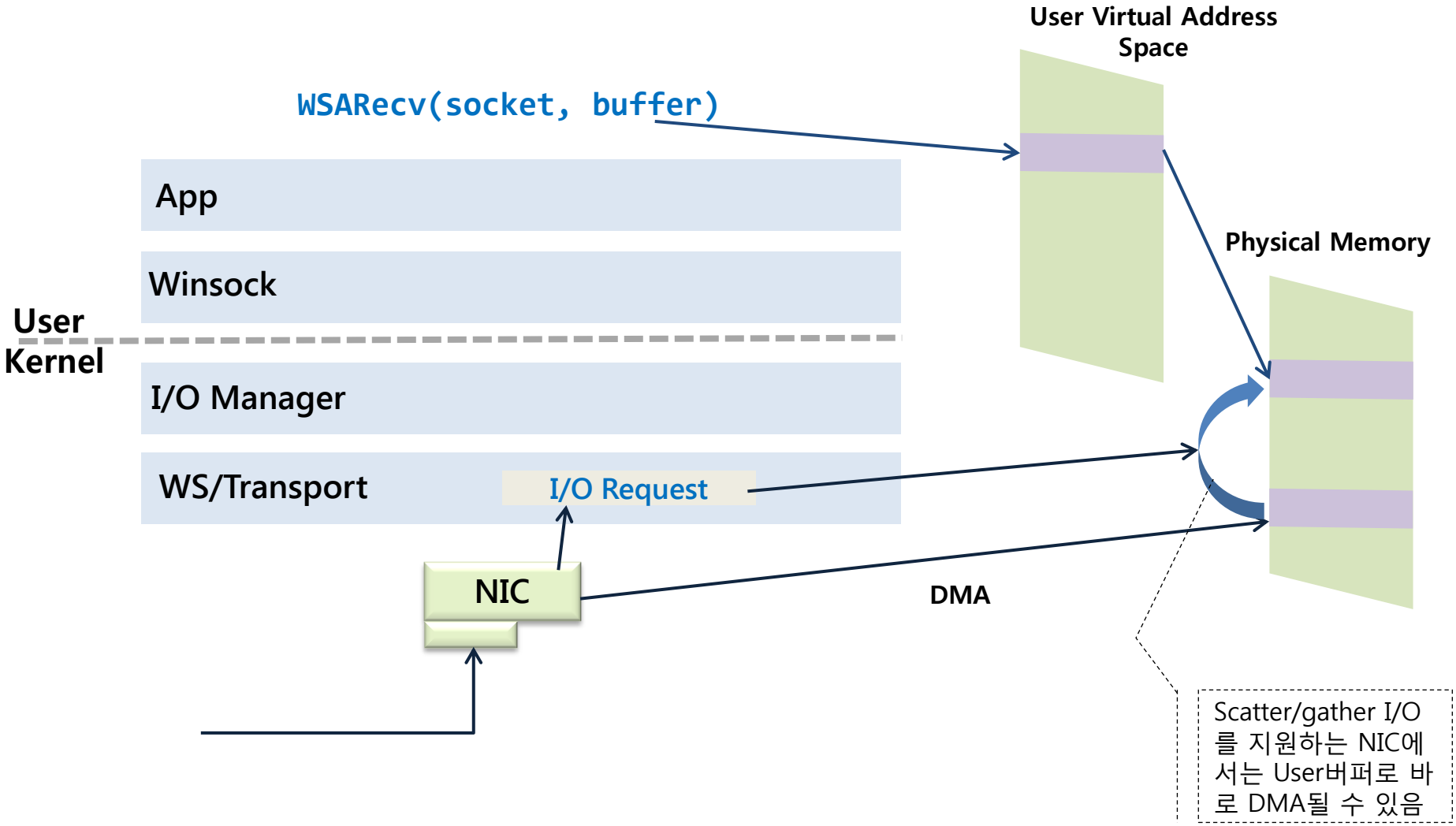
- I/O Initiation



IOCP 동작구조

NHN NEXT

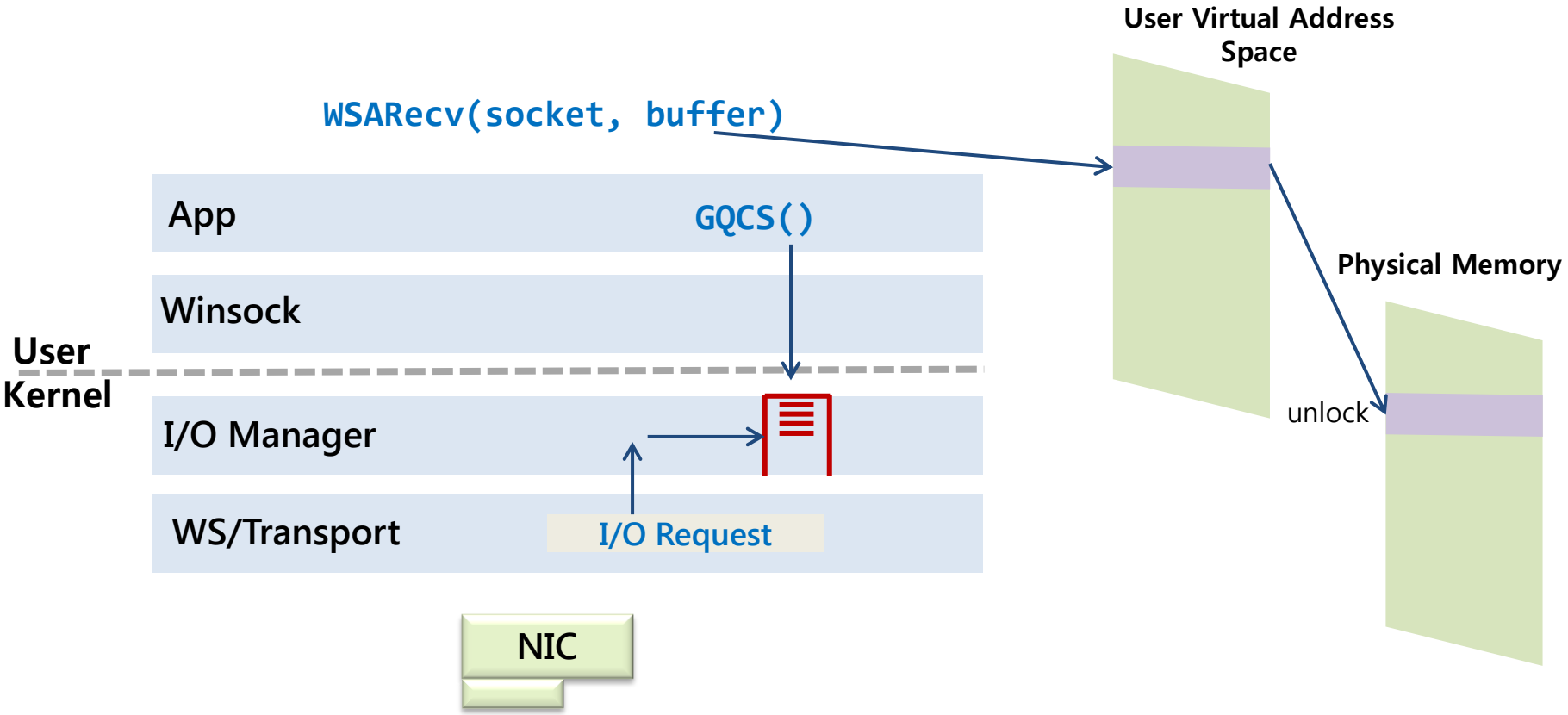
- I/O Processing



IOCP 동작구조

NHN NEXT

- I/O Completion



RIO 개요

NHN NEXT

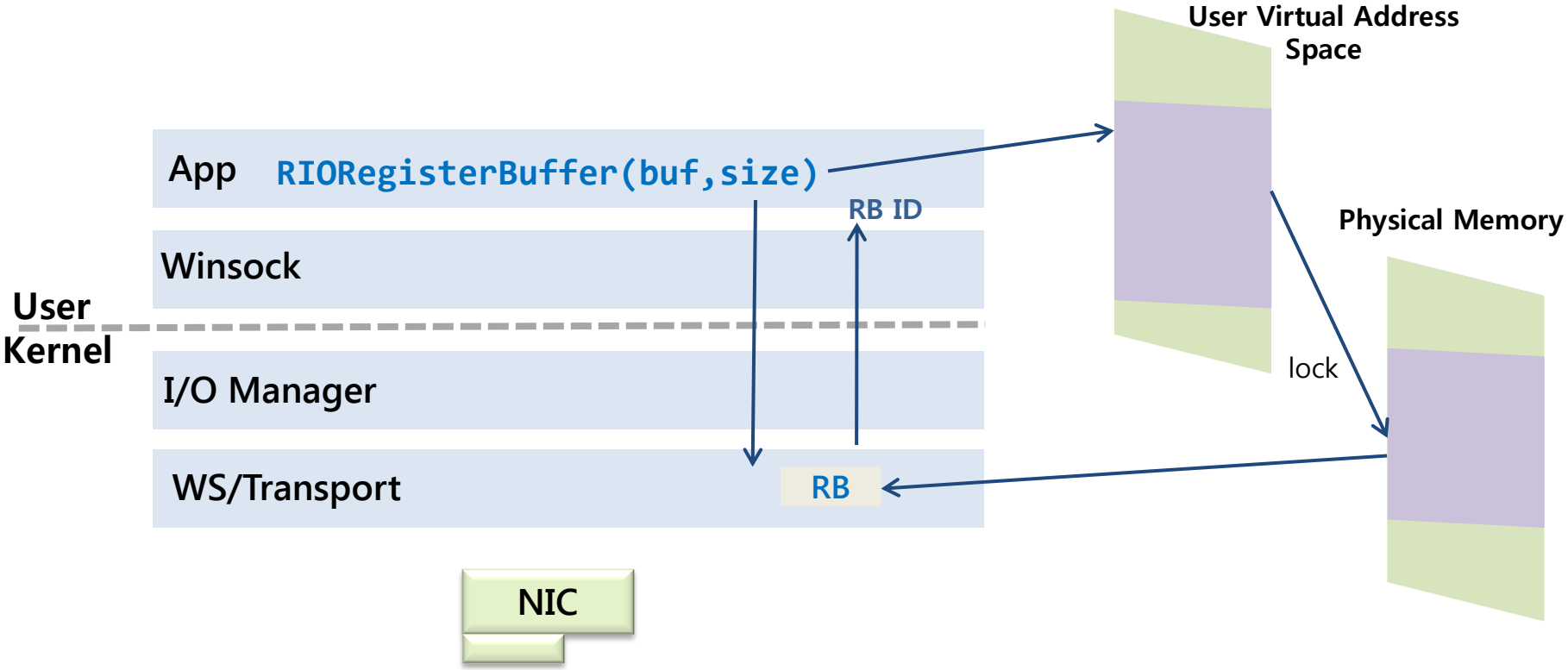
- Lower latency and jitter를 위해
  - 지연의 최소화 및 튀지 않는(?) 안정성 (예측가능성)
  - (예) 북미의 주식시세 정보 방송: 초당 5백만 업데이트 필요
  - (예) Database 서버나 UDP 스트리밍 같은 곳에서는 초당 패킷 처리량이 높을수록 최고의 성능을 뽑음
    - 엄청난 수의 작은 패킷 처리에 유리함 (by MS)

- 특징
  - I/O에 사용할 고정 크기의 버퍼를 등록하는 개념
    - 물리 메모리에 필요한 버퍼를 항상 pin해놓고 쓰기 때문에 매번의 I/O마다 page-lock/unlock이 없음
    - 메모리 사용량과 CPU사용량간의 Trade-off
    - I/O 버퍼 핸들링을 실제 I/O에서 분리하여 I/O 비용 감소 시킴
  - RIO에서는 커널 소켓 버퍼는 의미 없음
    - SO_SNDBUF, SO_RCVBUF

RIO Buffer 등록

NHN NEXT

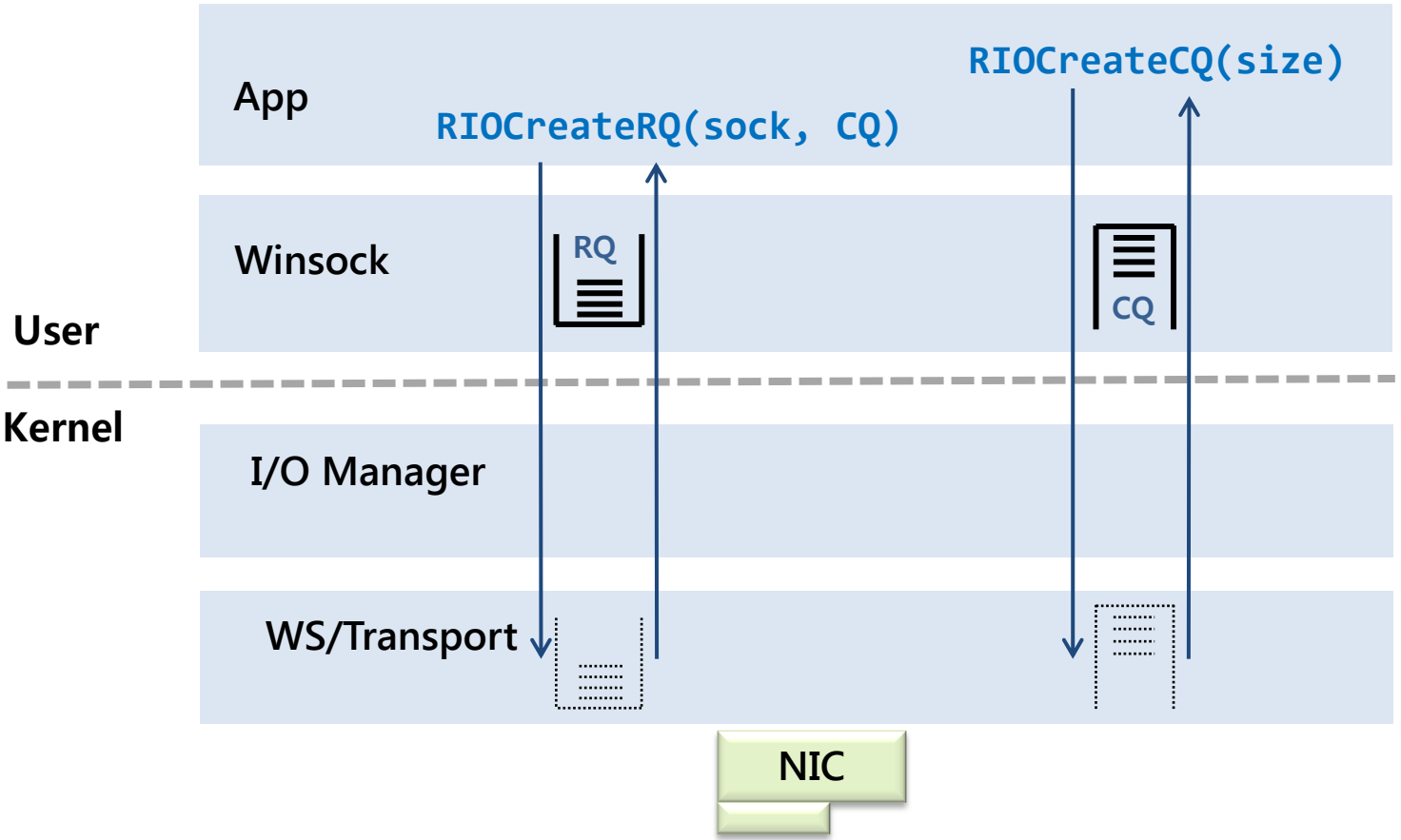
- RIO에서 사용할 고정 버퍼 등록 과정
  - RB: User영역과 Kernel영역이 공유하는 페이지로 PIN됨
    - RIODeregisterBuffer하기 전까지 계속 locking



RIO RQ/CQ 등록

NHN NEXT

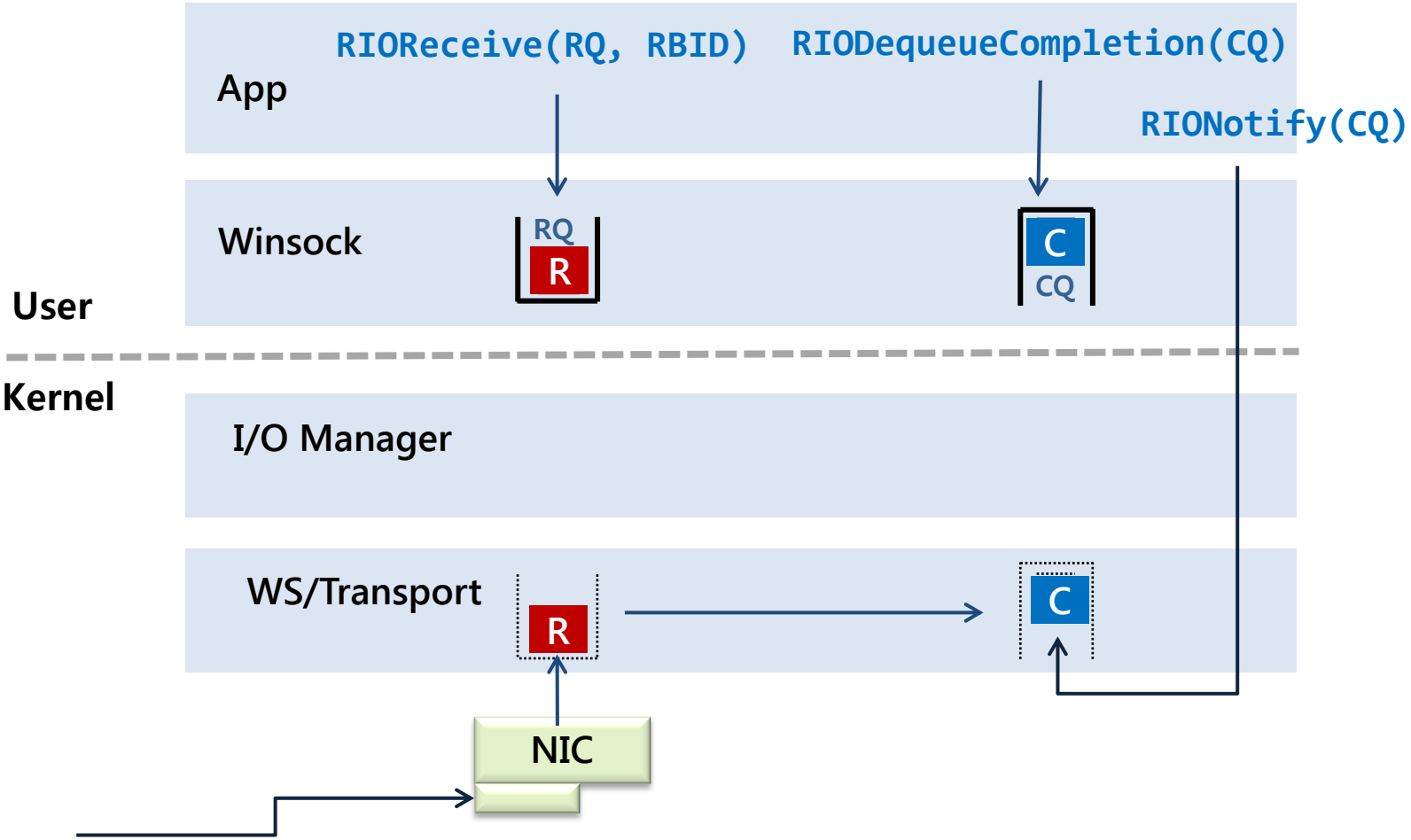
- RIO에서 사용할 고정 크기의 RQ 및 CQ 등록
  - 모든 I/O요청은 RequestQueue (RQ)를 통해 이루어짐
  - I/O 완료에 대한 통지는 CompletionQueue (CQ)를 통해 처리



RIO 동작구조

NHN NEXT

- I/O 처리 과정
  - DequeueCompletion시 복수개의 I/O완료 통지가 옴
    - 한번의 과정으로 CQ에 큐잉되어 있던 여러 개의 I/O 처리



성능 평가

NHN NEXT

- MS의 성능평가
  - RIO 사용시 Latency가 15~30% 가량 줄어들었다고 함
  - RIO 사용시 Throughput이 최대 2배 되었다고 함 (Datagrams/s)

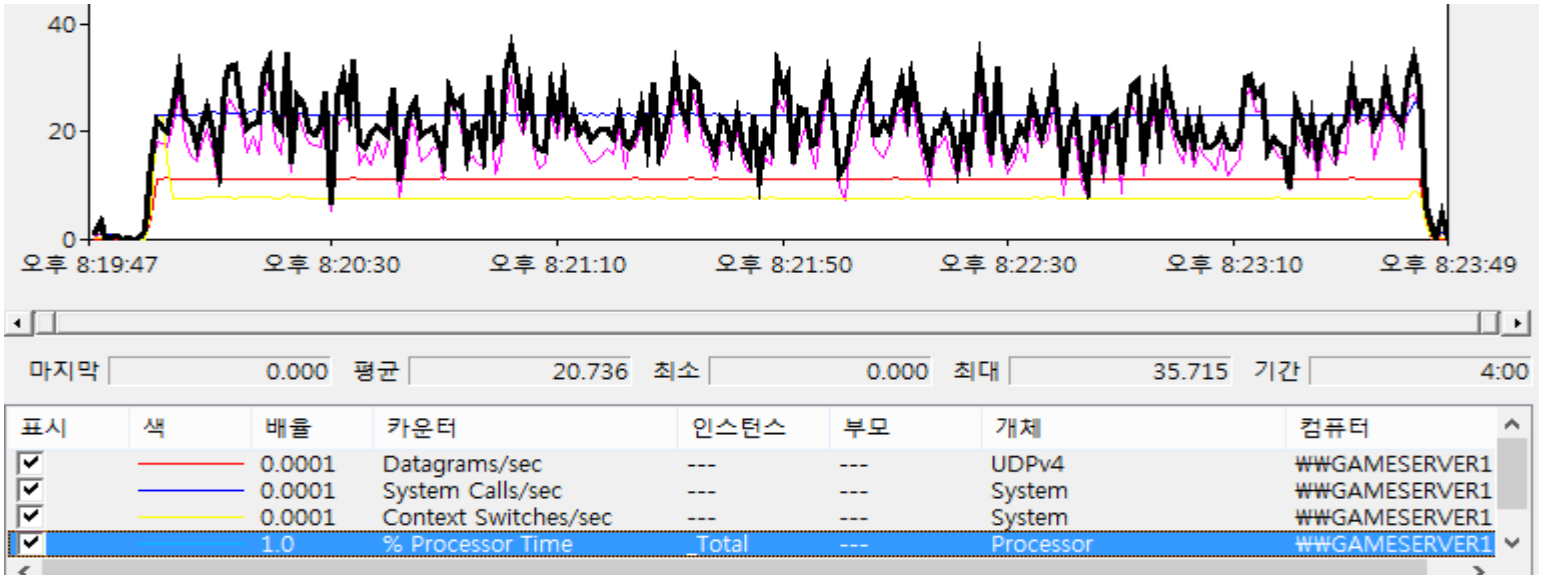
- 직접 해본 성능평가 (IOCP vs RIO)
  - 사용 장비
    - 클라이언트: i7-4770k, 16GB RAM, 1Gbps LAN, Windows 8
    - 서버: Mac-mini server 2012 late, Windows Server 2012
  - 1024 byte UDP 패킷 5천만개 전송 테스트
    - IOCP와 RIO의 경우 모두 1Gbps 대역 full로 활용함
      - 그러다보니 두 경우 모두 throughput과 UDP 드랍률이 비슷
        - » 즉, 의미 있는 결과 못냄 (10Gbps 대역에서는 차이가 많이 날 듯?)
  - 그러나, 같은 상황에서
    - RIO가 CPU사용률 약 2배 낮았음
    - RIO의 context-switches/sec가 6배 가량 낮았음

CPU Usage

NHN NEXT

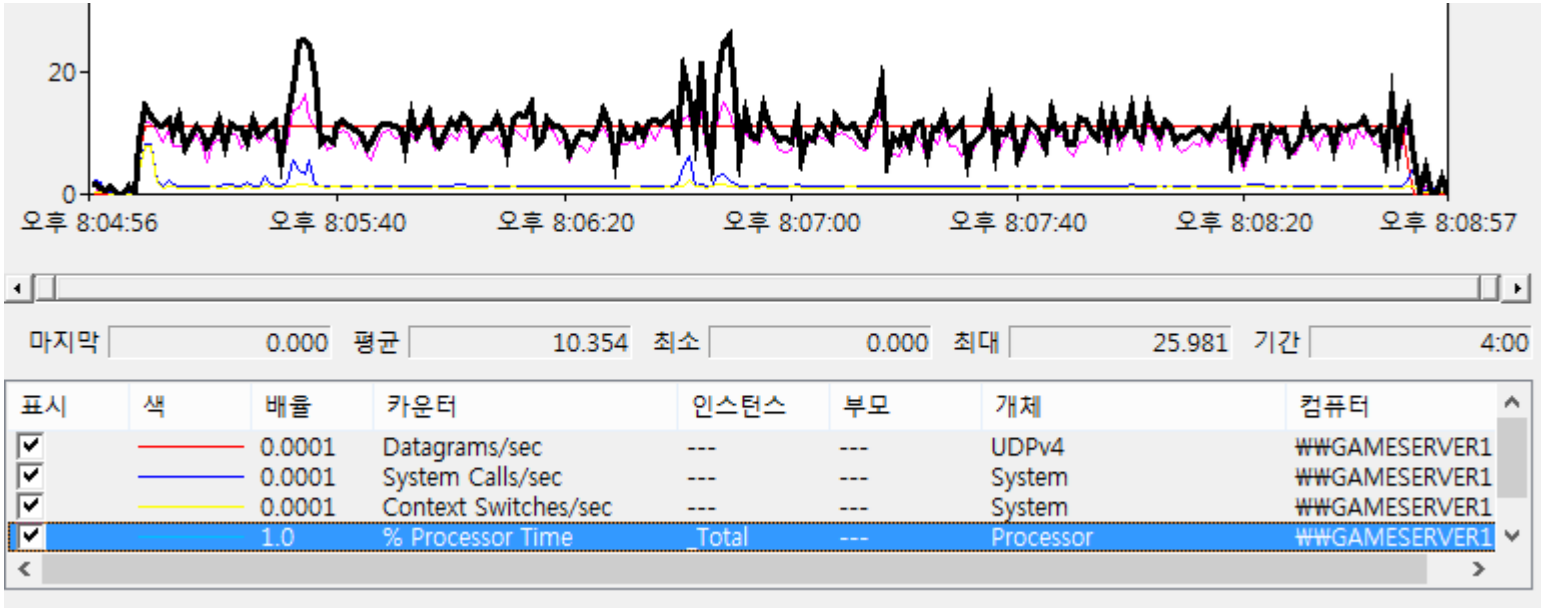
IOCP

약 20%



RIO

약 10%

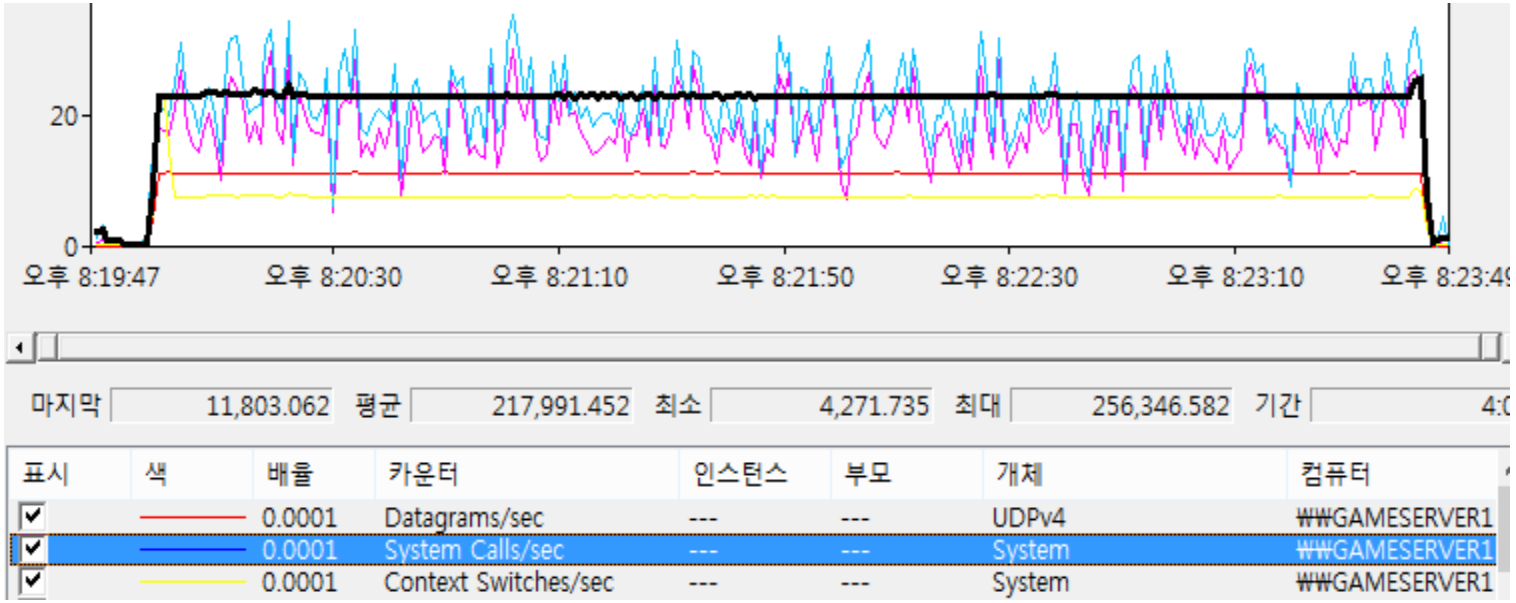


System Calls Per Second

NHN NEXT

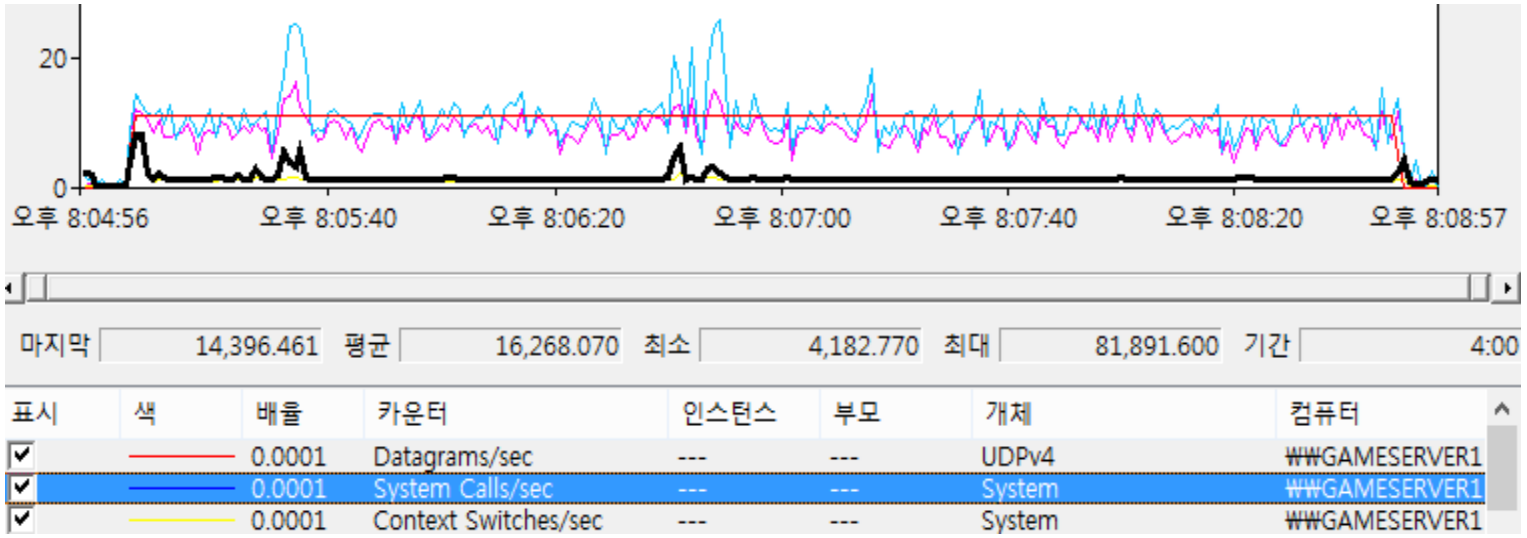
IOCP

217,991,452



RIO

16,268,070

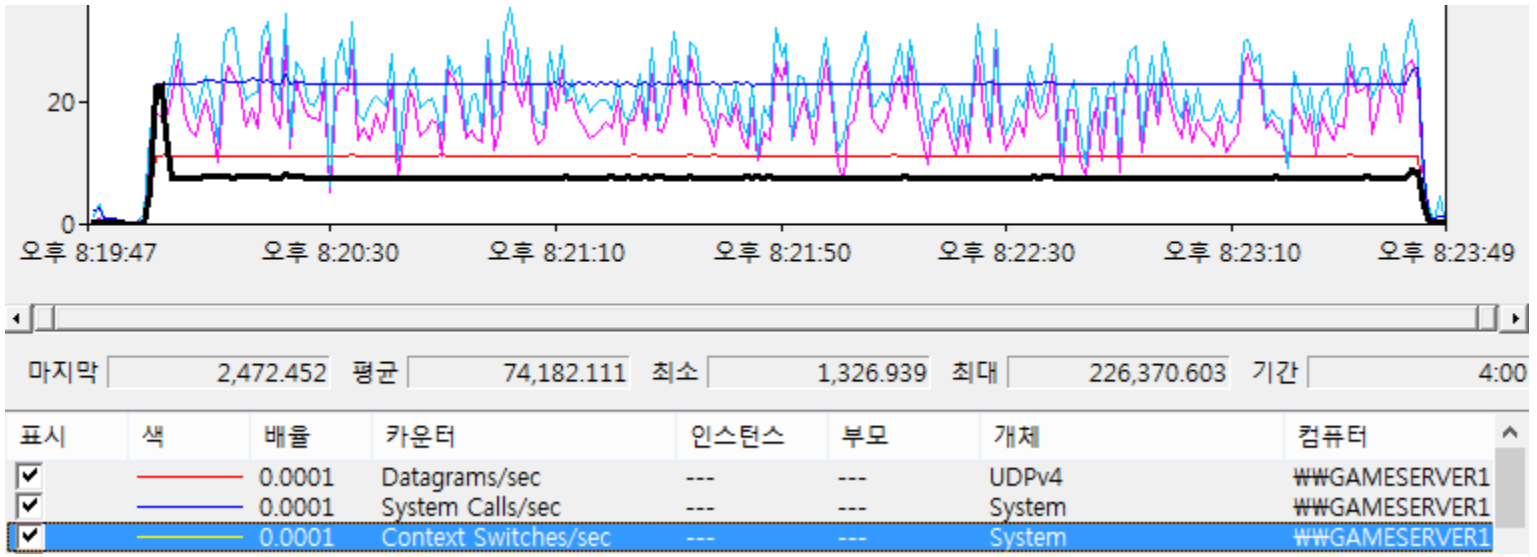


Context Switches Per Second

NHN NEXT

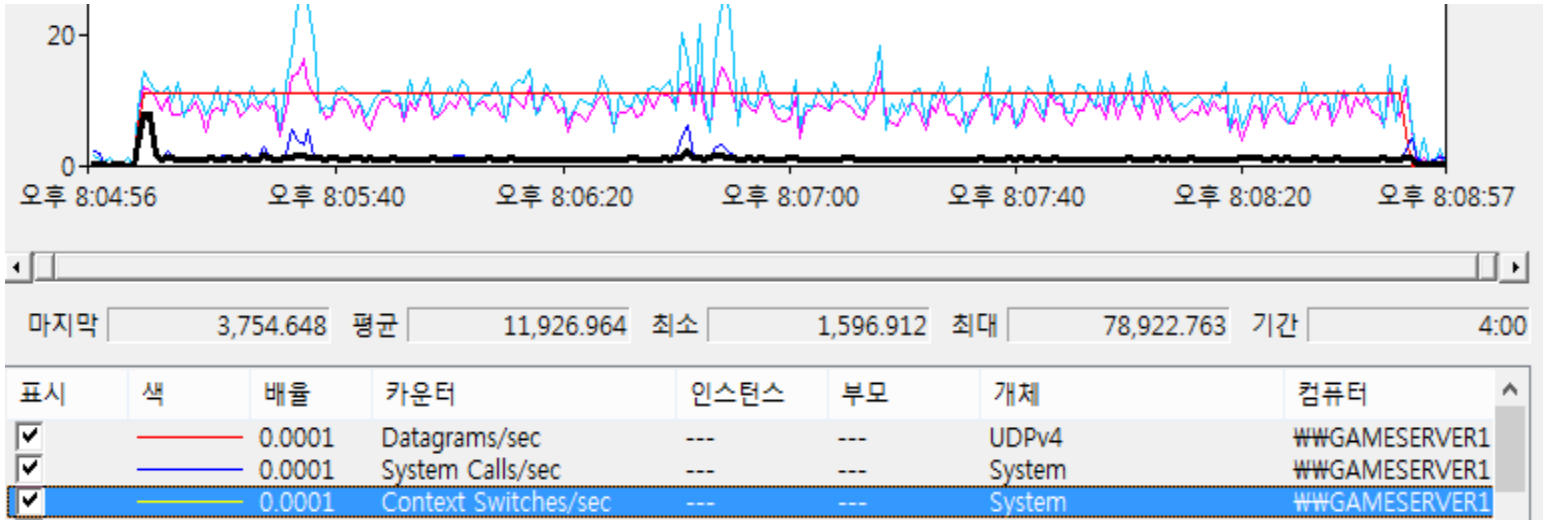
IOCP

74,182,111



RIO

11,926,964



결론?

NHN NEXT

- RIO가 성능은 확실히 좋음
  - Throughput: MS에 의하면 2배 가량 좋아진다고 함
  - CPU 사용률 등: 직접 성능 테스트 결과 월등히 좋음
    - Context-switching 및 System call 횟수도 훨씬 낮음

- 그런데…
  - 굳이 고성능 게임 서버용으로 RIO까지 쓸 필요가 있을까?
    - 점점 좋아지는 머신 성능, 더 복잡한 코딩 방법, MS 플랫폼 종속, …
    - Gigabit 네트워크 상에서의 throughput의 경우에
      - IOCP 뿐만 아니라 Linux의 EPOLL과도 별 차이 없었음
    - 게임 서버당 Gigabit 이상의 대역이 필요한 경우 다시 생각해봐도?
      - 트래픽이 많은 게임서버라고 해봤자 300~400Mbps 정도
      - 이 정도의 대역을 사용하는 경우, CPU 사용률의 차이도 적어짐

Sample Code

NHN NEXT

- 어디에도 제대로 된 샘플 코드가 없어서 직접 구현
  - MSDN 문서 조차도 제대로 되어 있지 않음
    - 수많은 삽질...
  - 최소한의 동작을 위한 코드만 들어 있음
  - 버그에 대한 책임 없음
- Source Code: RIO Echo Server
  - TCP, RIO only
    - https://github.com/zeliard/RIOTcpServer
  - UDP, IOCP+RIO
    - https://github.com/zeliard/RIOEchoServer

[Update] RIO 사용시 주의할 점

NHN NEXT

- RIO의 이벤트 통지
  - IOCP 또는 윈도우 Event를 이용하여 I/O 이벤트 통지를 받을 수 있지만 추천하지 않음 → 시스템 콜 횟수가 늘어나서 성능 저하
    - 앞의 성능 평가는 IOCP를 이벤트 통지 모델로 사용한 경우였음
      - 이렇게 해도 컨텍스트 스위칭이 6배 가량 낮았지만,
      - IOCP와 같은 이벤트 통지 없이 순수 RIODequeueCompletion만을 이용하는 것이 최고의 성능을 냄. 단, Sleep or WaitableTimer등을 이용하여 CPU 사용률 낮추는게 필요 (TCP버전의 소스코드 참고)

- Request/Completion Queue
  - Thread-safe하지 않기 때문에 정교한 설계가 필요함
    - (예) 세션 별로 특정 전담 thread 할당
  - CQ는 thread별로, RQ는 socket별로 만드는 것이 구조 및 효율면에서 좋음

- NUMA 장비에서 RIO
  - RIO가 user-level communication 방식인 관계로 타 CPU의 코어를 골고루 사용하는데 한계가 있음

참고 자료

NHN NEXT

- Tech to Develop Low Latency Apps, Build 2011.
  - http://channel9.msdn.com/Events/Build/BUILD2011/SAC-593T/player?w=960&h=544

- The Server Framework
  - http://www.serverframework.com/asynchronousevents/rio/